JOHN CANOSA

feature

Fundamentals of Firewire

The IEEE 1394 protocol (or Firewire, which is Apple's trademarked term) is one of the emerging bus protocols that will be important components of the connected future. Here's how it works.

People are sharing video, still images, and audio, and are constantly searching for faster, easier ways of transferring such information. This phenomenon is driving the convergence of computers, consumer equipment, and communications. Communication is the force that draws these separate market segments together.

Convergence will happen when seamless, high-speed communication becomes readily available. The IEEE 1394 protocol appears to be a strong contender for the communications channel that will make this happen.

The IEEE 1394-1995 protocol had its genesis at Apple Computer, which still retains the Firewire trademark. The goal of the protocol is to provide easy-to-use, low-cost, high-speed communications. The protocol is also very scaleable, provides for both asynchronous and isochronous applications, allows for access to vast amounts of memory mapped address space, and—perhaps most important for the aforementioned convergenceag—allows peer-to-peer communication.

Some people see 1394 and USB as competitors for the communications channel of the future, but in reality they are more complementary than competitive. USB is a lower-speed, lower-cost, host-based protocol. While 1394 and USB may compete in some mid-range applications, Table 1 illustrates how they will typically play in different spaces. The proposed upgrade of USB to 120Mbps and 240Mbps may alter this situation slightly, but not as much as some have predicted.

Confusion sometimes surrounds the alphabet soup that seems to envelop the 1394 protocol. The only currently approved specification is the IEEE 1394-1995 specification, which will be the basis for future extensions and enhancements. IEEE 1394-1995 supports transfer rates of 100, 200, and 400Mbps. As with many first cuts at a standard, 1394-1995 left some things up to the interpretation of the specification's implementers, which caused some interoperability problems and has led to the work being done on the 1394a specification. This revision provides some clarification on the original specification, changes some optional portions of the spec to mandatory, and adds some performance enhancements. The 1394a specification is near-

Convergence will happen when seamless, high-speed communication becomes readily available. This protocol appears to be a strong contender for the communications channel that will make this happen.

ing completion and should be approved in the near future; some semiconductor vendors, in fact, are already claiming compliance to the new specification. In addition to the 1394a specification, work is progressing on the 1394b specification. 1394b will provide for additional data rates of 800, 1,600, and 3,200Mbps. It will also provide for long-haul transmissions via both twisted pair and fiber optics, and offer backward compatibility with the existing standard.

This article covers the 1394-1995 standard and will speak to some of the enhancements in the 1394a revision. Details of the 1394b protocol will be left for a future article, when the specification is more firm.

Topology

The 1394 protocol is a peer-to-peer network with a point-to-point signaling environment. Nodes on the bus may have several ports on them. Each of these ports acts as a repeater, retransmitting any packets received by other ports within the node. Figure 1 shows what a typical consumer may have attached to their 1394 bus.

Because 1394 is a peer-to-peer protocol, a specific host isn't required, such as the PC in USB. In Figure 1, the digital camera could easily stream data to both the digital VCR and the DVD-RAM without any assistance from other devices on the bus.

Configuration of the bus occurs automatically whenever a new device is plugged in. Configuration proceeds from *leaf nodes* (those with only one other device attached to them) up through the branch nodes. A bus that has three or more devices attached will typically, but not always, have a branch node become the root node. I'll discuss configuration in more detail later in this article.

A 1394 bus appears as a large memory-mapped space with each node occupying a certain address range. The memory space is based to the IEEE 1212 Control and Status Register (CSR) Architecture with some extensions specific to the 1394 standard. Each node supports up to 48 bits of address space (256 TeraBytes). In addition, each bus can support up to 64 nodes, and the 1394 serial bus specification supports up to 1,024 buses. This gives a grand total of 64 address bits, or support for a whopping total of 16 ExaBytes of memory space—enough for the latest version of your favorite word processor and perhaps even a file or two!

Transfers and transactions

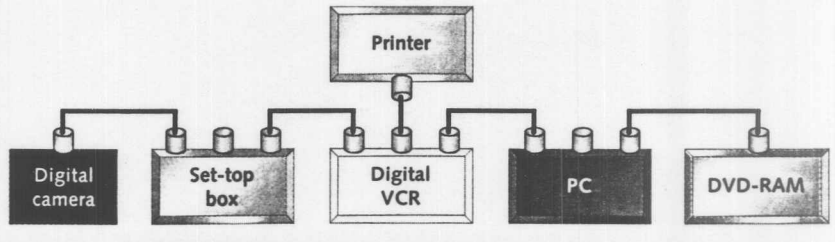
The 1394 protocol supports both asynchronous and isochronous data transfers, as I'll discuss in the following paragraphs.

Isochronous transfers. Isochronous transfers are always broadcast in a one-to-one or one-to-many fashion. No error correction nor retransmission is available for isochronous transfers. Up to 80% of the available bus bandwidth can be used for isochronous transfers. The delegation of bandwidth is tracked by a node on the bus that occupies the role of isochronous resource manager. This may or may not be the root node or the bus man-

TABLE 1 IEEE 1394 and USB market segmentation

	Performance	Applications	Attributes
USB Focus	Low speed 10Kbps to 100Kbps	Interactive devices (Game, VR)	Very low cost Ease of use Dynamic attach Multiple peripherals
	Medium speed 500Kbps to 10Mbps	ISDN, POTS, PBX, audio, limited video, bulk transfer	Low cost Guaranteed latency Higher bandwidth Ease of use
1394 Focus	High speed	Video, disk, LAN	High bandwidth Very low latency Ease of use

FIGURE 1 A Firewire bus



ager. The maximum amount of bandwidth an isochronous device can obtain is only limited by the number of other isochronous devices that have already obtained bandwidth from the isochronous resource manager.

Asynchronous transfers. Asynchronous transfers are targeted to a specific node with an explicit address. They are not guaranteed a specific amount of bandwidth on the bus, but they are guaranteed a fair shot at gaining access to the bus when asynchronous transfers are permitted. The maximum data block size for an asynchronous packet is determined by the transfer rate of the device, as specified in Table 2.

Asynchronous transfers are acknowledged and responded to. This allows error-checking and retransmission mechanisms to take place.

The bottom line is that if you're sending time-critical, error-tolerant data, such as a video or audio stream, isochronous transfers are the way to go. If the data isn't error-tolerant, such as a disk drive, then asynchronous transfers are preferable.

TABLE 2 Minimum data block size for an asynchronous packet

Cable Speed	Maximum Data Size
100Mbps	512 bytes
200Mbps	1,024 bytes
400Mbps	2,048 bytes

Physical layer

The 1394 specification defines four protocol layers, known as the physical layer, the link layer, the transaction layer, and the serial bus management layer. The layers are illustrated in Figure 3.

The physical layer of the 1394 protocol includes the electrical signaling, the mechanical connectors and cabling, the arbitration mechanisms, and the serial coding and decoding of the data being transferred or received. The cable media is defined as a three-pair shielded cable. Two of the pairs are used to transfer data, while the third pair provides power on the bus. The connectors are small six-pin devices, although the 1394a also defines a four-pin connector for self-powered leaf nodes. The power signals aren't provided on the four-pin connector. The baseline cables are limited to 4.5m in length. Thicker cables allow for longer distances.

The two twisted pairs used for signaling, called out as TPA and TPB, are bidirectional and are tri-state capable. TPA is used to transmit the strobe signal and receive data, while TPB is used to receive the strobe signal and transmit data. The signaling mechanism uses data strobe encoding, a rather clever tech-

FIGURE 2 IEEE-1394 address space

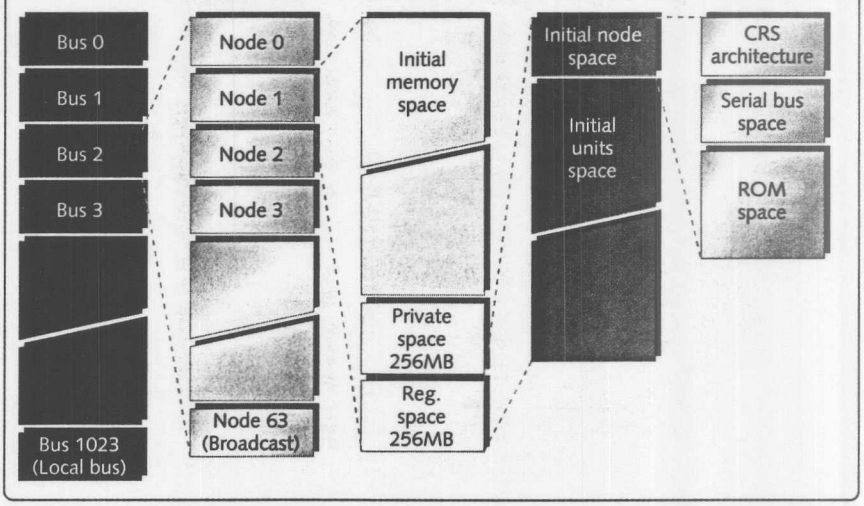
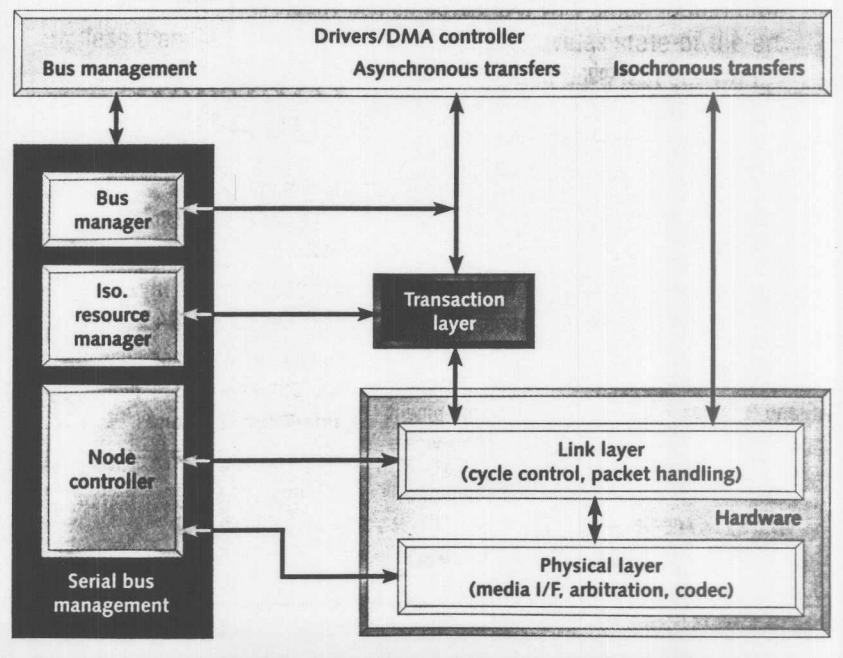


FIGURE 3 IEEE-1394 protocol layers



nique that allows easy extraction of a clock signal with much better jitter tolerance than a standard clock/data mechanism. With data strobe encoding, either the data or the strobe signal (but not both of them) change in a bit cell. Data strobe encoding is shown in Figure 4.

Configuration

The physical layer plays a major role in the bus configuration and normal arbitration phases of the protocol. Configuration consists of taking a relatively flat physical topology and turning it into a logical tree structure with a root node at its focal point. A bus is reset and reconfigured whenever a device is added or removed. A reset can also be initiated via software. Configuration consists of bus reset and initialization, tree identification, and self identification.

Reset. Reset is signaled by a node driving both TPA and TPB to logic 1. Because of the "dominant 1s" electrical definition of the drivers, a logic 1 will always be detected by a port, even if its bidirectional driver is in the transmit state. When a node detects a reset condition on its drivers, it will propagate this signal to all of the other ports that this node supports. The node then enters the idle state for a given period of time to allow the reset indication to propagate to all other nodes on the bus. Reset clears any topology information within the node, although isochronous resources are "sticky" and will tend to remain the same during resets.

Tree identification. The tree identification process defines the bus topology. Let's take the example of our sample home consumer network shown in Figure 1. After reset, but before tree identification, the bus has a flat logical topology that maps directly to the physical topology. After tree identification is complete, a single node has gained the status of root node. The tree identification proceeds as follows.

After reset, all leaf nodes present a Parent_Notify signaling state on their data and strobe pairs. Note that this is a signaling state, not a transmitted packet. The whole tree identification process occurs in a matter of microseconds. In our example, the digital camera will signal the set-top box, the printer will signal the digital VCR, and the DVD-RAM will signal the PC. When a branch node receives the Parent_Notify signal on one of its ports, it marks that port as containing a child, and outputs a Child_Notify signaling state on that port's data and strobe pairs. Upon detecting this state, the leaf node marks its port as a parent

port and removes the signaling, thereby confirming that the leaf node has accepted the *child* designation. At this point our bus appears as shown in Figure 5. The ports marked with a "P" indicate that a device which is closer to the root node is attached to that port, while a port marked with a "C" indicates that a node farther away from the root node is attached. The port numbers are arbitrarily assigned during design of the device and play an important part in the self identification process.

FIGURE 4 Data strobe encoding

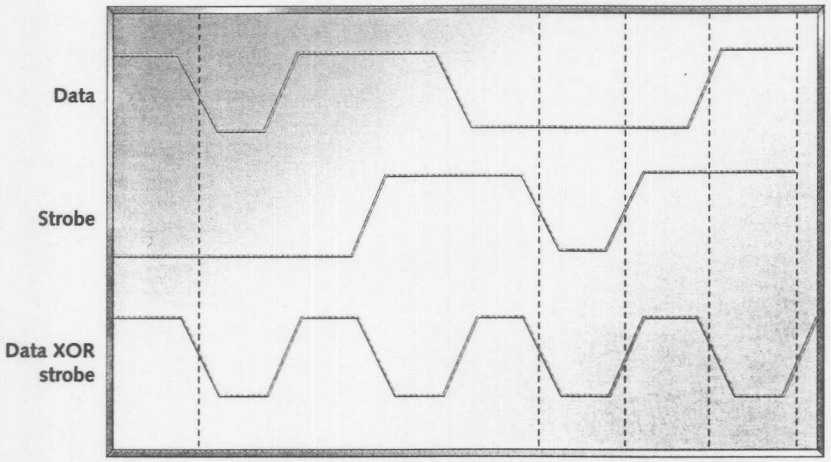


FIGURE 5 Bus after leaf node identification

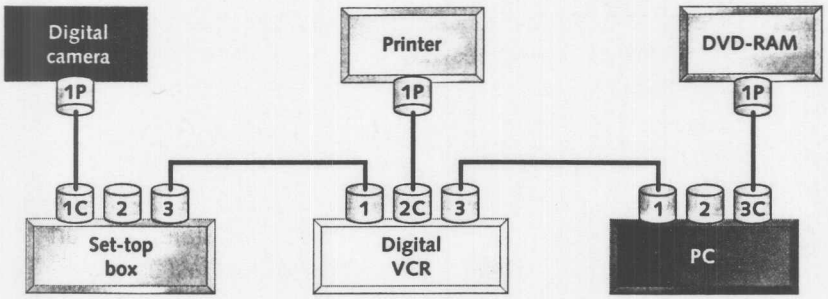
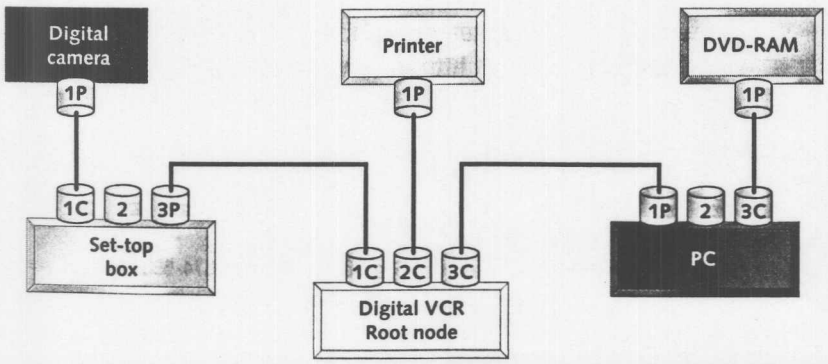


FIGURE 6 Bus after tree identification is complete



After the leaf nodes have identified themselves, the digital VCR still has two ports that have not received a Parent_Notify, while the set-top box and the PC branch node both have only one port with an attached device that has not received a Parent_Notify. Therefore, both the set-top box and the PC start to signal a Parent_Notify on the one port that has not yet received one. In this case, the VCR receives the Parent_Notify on both of its remaining ports, which it acknowledges with a Child_Notify condition. Because the VCR has marked all of its ports as children, the VCR becomes the root node. The final configuration is shown in Figure 6.

Note that two nodes can be in contention for root node status at the end of the process. In this case, a random back-off timer is used to eventually settle on a root node. A node can also force itself to become root node by delaying its participation in the tree identification process for a while. See References 1 and 2 for more details.

Self identification. Once the tree topology is defined, the self identification phase begins. Self identification consists of assigning physical IDs to each node on the bus, having neighboring nodes exchange transmission speed capabilities, and making all of the nodes on the bus aware of the topology that exists. The self identification phase begins with the root node sending an arbitration grant signal to its lowest numbered port. In our example, the digital VCR is the root node and it signals the set-top box. Since the set-top box is a branch node, it will propagate the Arbitration Grant signal to its lowest numbered port with a child node attached. In our case, this

port is the digital camera. Because the digital camera is a leaf node, it cannot propagate the arbitration grant signal downstream any farther, so it assigns itself physical ID 0 and transmits a self ID packet upstream. The branch node (set-top box) repeats the self ID packet to all of its ports with attached devices. Eventually the self ID packet makes its way back up to the root node, which proceeds to transmit the self ID packet down to all devices on its higher-numbered ports. In this manner, all attached devices receive the self ID packet that was transmitted by the digital camera. Upon receiving this packet, all of the other devices increment their self ID counter. The digital camera then signals a *self ID done* indication upstream to the set-top box, which indicates that all nodes attached downstream on this port have gone through the self ID process. Note that the set-top box does *not* propagate this signal upstream toward the root node because it hasn't completed the self ID process.

The root node will then continue to signal an Arbitration Grant signal to its lowest numbered port which in this case is still the set-top box. Because the set-top box has no other attached devices, it assigns itself physical ID 1 and transmits a self ID packet back upstream. This process continues until all ports on the root node have indicated a *self ID done* condition. The root node then assigns itself the next physical ID. The root node will always be the highest-numbered device on the bus. If we follow through with our example, we come up with the following physical IDs: digital camera = 0; set-top box = 1; printer = 2; DVD-RAM = 3; PC = 4; and the digital VCR, which is the root node, 5.

Note that during the self ID process, parent and children nodes are also exchanging their maximum speed capabilities. This process also exposes the Achilles heel of the 1394 protocol. Nodes can only transmit as fast as the slowest device between the transmitting node and the receiving node. For example, if the digital camera and the digital VCR are both capable of transmitting at 400Mbps, but the set-top box is only capable of transmitting at 100Mbps, the high-speed devices cannot use the maximum rate to communicate amongst themselves. The only way around this problem is for the end user to reconfigure the cabling so the low-speed set-top box is not physically between the two high-speed devices.

Also during the self ID process, all nodes wishing to become the isochronous resource manager will indicate this fact in their self ID packet. The

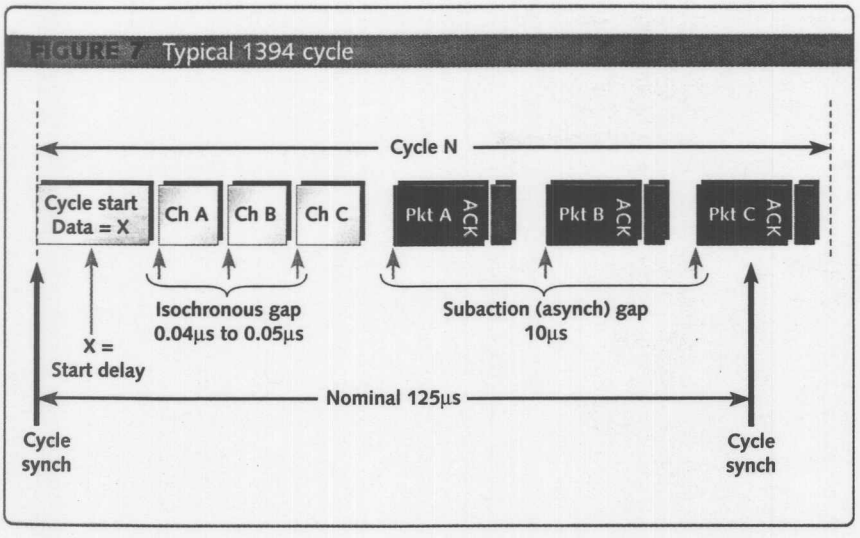


FIGURE 7 Typical 1394 cycle

highest numbered node that wishes to become resource manager will receive the honor.

Normal arbitration

Once the configuration process is complete, normal bus operations can begin. To fully understand arbitration, a knowledge of the cycle structure of 1394 is necessary.

A 1394 cycle is a time slice with a nominal 125µs period. The 8kHz cycle clock is kept by the cycle master, which is also the root node. To begin a cycle, the cycle master broadcasts a cycle start packet, which all other devices on the bus use to synchronize their time-bases.

Immediately following the cycle start packet, devices that wish to broadcast their isochronous data may arbitrate for the bus. Arbitration consists of signaling your parent node that you wish to gain access to the bus. The parent nodes in turn signal their parents and so on, until the request reaches the root node. In our previous example, suppose the digital camera and the PC wish to stream data over the bus. They both signal their parents that they wish to gain access to the bus. Since the PC's parent *is* the root node, its request is received first and it is granted the bus. From this scenario, it is evident that the closest device to the root node wins the arbitration.

Because isochronous channels can only be used once per cycle, when the next isochronous gap occurs, the PC will no longer participate in the arbitration. This condition allows the digital camera to win the next arbitration. Note that the PC could have more than one isochronous channel, in which case it would win the arbitration until it had no more channels left. This points out the important role of the isochronous resource manager: it will not allow the allotted isochronous channels to require more bandwidth than available.

When the last isochronous channel has transmitted its data, the bus becomes idle waiting for another isochronous channel to begin arbitration. Because there are no more isochronous devices left waiting to transmit, the idle time extends longer than the isochronous gap until it reaches the duration defined as the subaction (or asynchronous) gap. At this time, asynchronous devices may begin to arbitrate for the bus. Arbitration proceeds in the same manner, with the closest device to the root node winning arbitration.

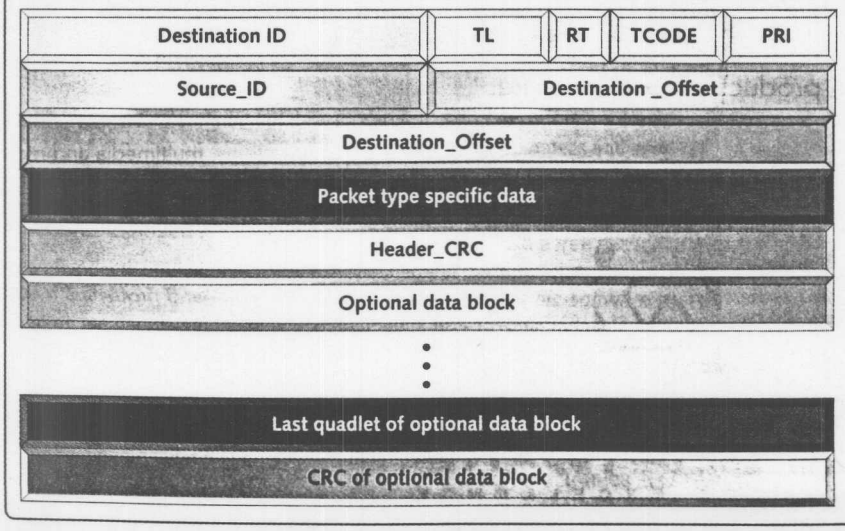
This point brings up an interesting scenario: because asynchronous

devices can send more than one packet per cycle, the device closest to the root node (or the root node itself) might be able to hog the bus by always winning the arbitration. This scenario is dealt with using what is called the *fairness interval* and the arbitration rest gap. The concept is simple—once a node wins the asynchronous arbitration and delivers its packet, it clears its arbitration enable bit. When this bit is cleared, the physical layer no longer participates in the arbitration process, giving devices farther away from the root node a fair shot at gaining access to the bus. When all devices wishing to gain access to the bus have had their fair shot, they all wind up having their arbitration enable bits cleared, meaning no one is trying to gain access to the bus. This causes the idle time on the bus to go longer than the 10µs subaction gap until it finally reaches 20µs, which is called the arbitration reset gap. When the idle time reaches this point, all devices may reset their arbitration enable bits and arbitration can begin all over again.

TABLE 3 Seventeen signals of the link layer to physical layer interface

Signal Name	Source	Description
LReq	Link layer	Link request—used to initiate a request to send a packet, as well as a request to read directly from a PHY register.
SClk	Physical layer	49.152MHz clock used to synchronize data readout. (The frequency may change depending on data rates with 1394b.)
Data[0:7]	Either	Data—higher transfer speeds use an increasing number of bits: 100Mbps —D[0:1] 200Mbps —D[0:3] 400Mbps —D[0:7] Note that the width of this data bus may expand to 16 bits with 1394b.
Ctl[0:1]	Either	Control interface—defines what state the interface is in.
LPS	Link layer	Link power status—indicates that the link layer controller is powered.
Link On	Physical layer	Indicates that the physical layer has been powered on.
Direct	Neither	Indicates that no isolation barrier exists.
Backplane	Physical layer	High if physical layer is a backplane implementation.
Clk25	Neither	Indicates that SClk is only 24.576MHz; valid in a backplane implementation only.

FIGURE 3 Asynchronous packet format



Link layer

The link layer is the interface between the physical layer and the transaction layer. The link layer is responsible for checking received CRCs and calculating and appending the CRC to transmitted packets. In addition, because isochronous transfers do not use the transaction layer, the link layer is directly responsible for sending and receiving isochronous data. The link layer also examines the packet header information and determines the type of transaction that is in progress. This information is then passed up to the transaction layer.

The interface between the link layer and the physical layer is listed as an informative (not required) appendix in the IEEE 1394-1995 specification. In the 1394a addendum, however, this interface becomes a required part of the specification. This change was instituted to promote interoperability amongst the various 1394 chip vendors.

The link layer to physical layer interface consists of a minimum of 17 signals that must be either magnetically or capacitively isolated from the PHY. These signals are defined in Table 3.

A typical link layer implementation has the PHY interface, a CRC checking and generation mechanism, transmit and receive FIFOs, interrupt registers, a host interface and at least one DMA channel.

Transaction layer

The transaction layer is used for asynchronous transactions. The 1394 protocol uses a request-response mechanism, with confirmations typically generated within each phase. Several types of transactions are allowed. They are listed as follows:

TABLE 4 Asynchronous packet descriptions

Name	Description
Destination_ID	The concatenation of the Bus and Node address of the intended node. All ones indicate a broadcast transmission.
TL	Transaction Label specified by the requesting node.This value is also used in the response packet.
RT	Retry Code that defines whether this is a retry and what retry mechanism is being used.
TCODE	Transaction Code defines the type of transaction (Read request, read response, Acknowledge, and so forth).
PRI	Priority—used only in backplane environments
Source_ID	Specifies Bus and Node that generated this packet.
Destination_Offset	The address location within the destination node that is being accessed.
Packet type Specific Data	Can indicate data length for block reads and writes, or contain actual data for a quadlet write request or quadlet read response.
Header_CRC	CRC Value for the Data
Optional Data	Quadlet aligned data specific to the type of the packet.
Optional Data CRC	CRC for the Optional Data

- Simple quadlet (four-byte) read
- Simple quadlet write
- Variable-length read
- Variable-length write
- Lock transactions

Lock transactions allow for atomic swap and compare and swap operations to be performed.

Asynchronous packets have a standard header format, along with an optional data block. The packets are assembled and disassembled by the link layer controller. Figure 8 shows the format of a typical asynchronous packet.

Transactions can be split, concatenated, or unified. Figure 9 illustrates a split transaction. The split transaction occurs when a device cannot respond fast enough to the transaction request. When a request is received, the node responds with an acknowledge packet. An acknowledge packet is sent after every asynchronous packet. In fact, the acknowledging device doesn't even have to arbitrate for the bus; control of the bus is automatic after receiving an incoming request or response packet.

In Figure 9, the responder node sends the acknowledge back and then prepares the data that was requested. While this is going on, other devices may be using the bus. Once the responder node has the data ready, it begins to arbitrate for the bus, to send out its response packet containing the desired data. The requester node receives this data and returns an acknowledge packet (also without needing to re-arbitrate for the bus).

If the responder node can prepare the requested data quickly enough, the entire transaction can be concatenated. This removes the need for the responding node to arbitrate for the bus after the acknowledge packet is sent.

For data writes, the acknowledgement can also be the response to the write, which is the case in a unified transaction. If the responder can accept the data fast enough, its acknowledge packet can have a transaction code of *complete* instead of *pending*. This eliminates the need for a separate response transaction altogether. Note that unified read and lock transactions aren't possible, and the acknowledge packet can't return data. Figure 10 shows the different types of transactions supported by 1394.

1394a arbitration enhancements

The 1394a addendum adds three new types of arbitration to be used with

asynchronous nodes: acknowledged accelerated arbitration, fly-by arbitration, and token-style arbitration.

Nodes on the bus must assume the roles of cycle master, isochronous resource manager, and bus manager.

Acknowledged accelerated arbitration. When a responding node also has a request packet to transmit, the responding node can immediately transmit its request without arbitrating for the bus. Normally the responding node would have to go through the standard arbitration process.

Fly-by arbitration. A node that contains several ports must act as a repeater on its active ports. A multiport node may use fly-by arbitration on packets that don't require acknowledgement (isochronous packets and acknowledge packets). When a node using this technique is repeating a packet *upstream* toward the root node, it may concatenate an identical speed packet to the end of the current packet. Note that asynchronous packets may not be added to isochronous packets.

Token-style arbitration. Token-style arbitration requires a group of cooperating nodes. When the cooperating node closest to the root node wins a normal arbitration, it can pass the arbitration grant down to the node farthest from the root. This node sends a normal packet, and all of the cooperating nodes can use fly-by arbitration to add their packets to the original packet as it heads upstream.

Bus management

Bus management on a 1394 bus involves several different responsibilities that may be distributed among more than one node. Nodes on the bus must assume the roles of cycle master, isochronous resource manager, and bus manager.

Cycle master. The cycle master initiates the 125µs cycles. The root node *must* be the cycle master; if a node that is not cycle master capable becomes root node, the bus is reset and a node that is cycle master capable is forced to be the root. The cycle master broadcasts a cycle start packet every 125µs. Note that a cycle start can be delayed while an asynchronous packet is being transmitted or acknowledged. The cycle master deals with this by including the amount of time that the cycle was delayed in the cycle start packet.

FIGURE 9 A split transaction

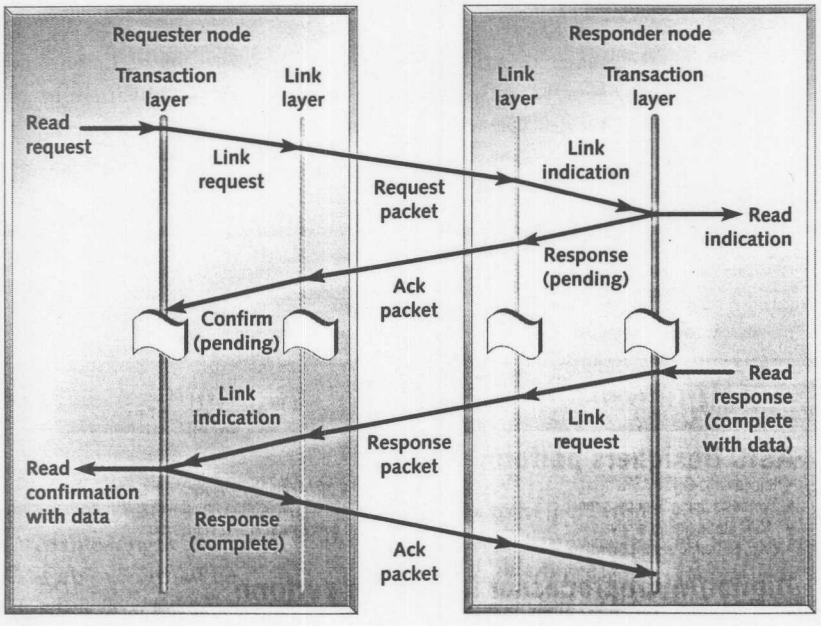
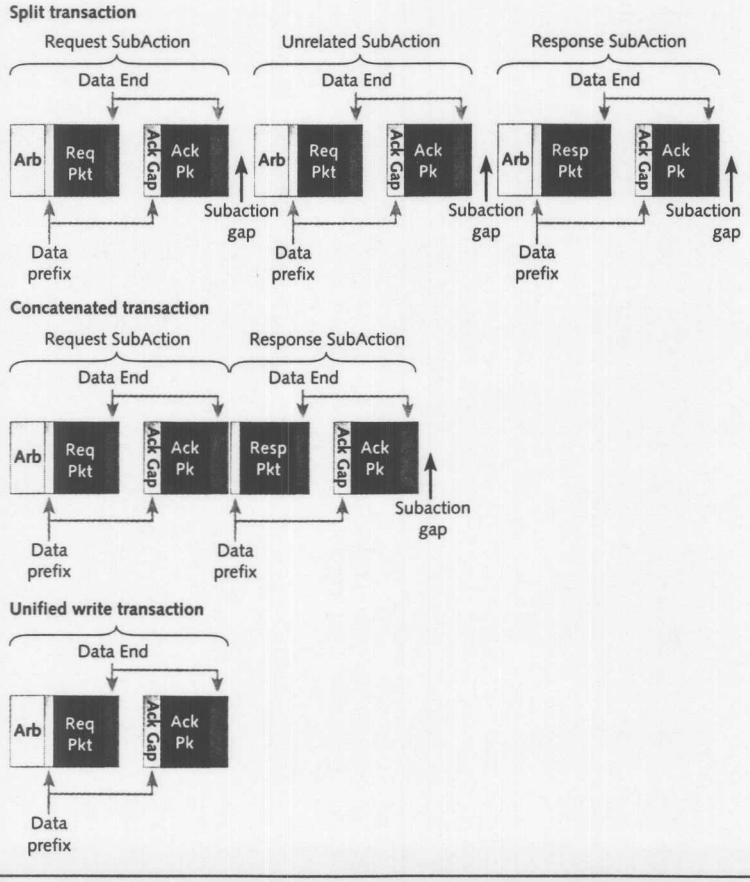


FIGURE 10 IEEE-1394 transaction types



Isochronous resource manager. The isochronous resource manager must be isochronous transaction capable. The isochronous resource manager must also implement several additional registers. These registers include the Bus Manager ID Register, the Bus Bandwidth Allocation Register, and the Channel Allocation Register. Isochronous channel allocation is performed by a node that wishes to transmit isochronous packets. These nodes must allocate a channel from the Channel Allocation Register by reading the bits in the 64-bit register. Each channel has one bit associated with it. A channel is available if its bit is set to a logic 1. The requesting node sets the first available channel bit to a logic 0 and uses this bit number as the channel ID.

In addition, the requesting node must examine the Bandwidth Available Register to determine how much bandwidth it can consume. The total amount of bandwidth available is 6,144 allocation units. One allocation unit is the time required to transfer one quadlet at 1,600Mbps. A total of 4,915 allocation units are available for isochronous transfers if any asynchronous transfers are used. Nodes wishing to use isochronous bandwidth must subtract the amount of bandwidth needed from the Bandwidth Available Register.

Bus manager. A bus manager has several functions, including publishing the topology and speed maps, managing power, and optimizing bus traffic. The topology map may be used by nodes with a sophisticated user interface that could instruct the end user on the optimum connection topology to enable the highest throughput between nodes. The speed map is used by nodes to determine what speed it can use to communicate with other nodes.

The bus manager is also responsible for determining whether the node that has become root node is cycle master capable. If it isn't, the bus man-

and physical layer controllers.

ager searches for a node that is cycle master capable and forces a bus reset that will select that node as root node. The bus manager might not always find a capable node; in this case, at least some of the bus management functions are performed by the isochronous resource manager.

support

Hardware. Several manufacturers offer components for engineers designing systems to support IEEE 1394. Integrated circuit providers typically provide a chipset that includes a link layer controller and a physical layer controller. One of the goals of the 1394a addendum is to provide interoperability among the various link layer and physical layer controllers. Some of the available ICs and cores are shown in Table 5.

Complete PCI-based cards that plug into a PC backplane are available from companies such as Adaptec, Sony, and Texas Instruments.

Software. IEEE 1394 is directly supported in the new Windows Driver Model (WDM), which is used in Windows 98 and will be available in Windows NT 5.0. For chipsets and devices to support the drivers provided in the new versions of Windows, several members of the 1394 Trade Association have banded together to create the 1394 Open Host Controller Interface (OHCI) Specification. The OHCI provides a link layer controller, as well as bus management functionality. In addition, the OHCI defines several DMA controllers for asynchronous and isochronous transactions. These controllers provide registers that a standard 1394 driver, provided by Microsoft, can use to configure the controller and schedule transactions.

Microsoft provides WDM streaming drivers to support audio and video devices such as DVD players, MPEG decoders, tuners, and audio codecs. These streaming drivers permit low-latency support for isochronous channels. The drivers minimize the transitions between user mode and kernel mode, which significantly reduces the overhead for driver calls and data movement.

For storage devices, printers, and scanners, Windows NT 5.0 supports the Serial Block Protocol (SBP-2).

The IEEE 1394 protocol, USB, Ethernet, and IrDA will be the data channels of the future. Any embedded system that needs to share information will use at least one of the aforementioned methods.

Microsoft recommends that devices be written to support the SCSI command set so the device can use the existing SCSI class driver that sits on top of the SBP-2 driver. If the vendor doesn't support the SCSI protocol, they will need to write their own class driver to support their own command set.

In addition to the SBP-2 specification for storage devices, other standard data formats that ride on top of 1394 are in various stages of completion. These include the Tailgate specification, which defines a method for adapting ATA/ATAPI controllers to 1394, a digital video (DV) standard, and a printer protocol. The Digital Still Image Working Group and an industrial control and instrumentation group are also working on related standards.

Embedded systems designers have also seen some RTOS vendors add support for 1394, including Integrated Systems and Wind River. These vendors typically support a third-party protocol stack that has been ported to their RTOS. Zayante, Award Software, and Technology Rendevous each have a 1394 stack that they claim is OS-independent. Windows CE doesn't currently have native support for 1394, but it will undoubtedly support it in the near future. Third-party support fills the existing gap.

TABLE 1: Some available ICs and cores

Link Layer Controllers

Manufacturer	Part Number	Description
Fujitsu Microelectronics	MB8661x	Combined link/PHY core & ICs
IBM	IBM21S650PFA	PCI-based link layer controller
	IBM21S550PFB	Generic bus interface link layer controller
Innovative Semiconductor	SL75x	Link layer cores
Philips Semicondutor	PDI1394L11	A/V link layer controller
Phoenix Technologies	VirtualLink 1394a	Compatible link layer cores
Sand	1394 Device Controller	1394 link layer core
Symbios	SYM13FW600	PCI bus interface link layer
	SYM13FW500	1394-to-ATA/ATAPI interface
NEC	uPD728xx	OHCI link layer IC (some integrate PHY)
Texas Instruments	TSB12LV21B	LynxHCI (PCI) IC
	TSB12LV22	OHCI (PCI) IC
	TSB12LV31	General-purpose bus interface IC
Sony	CXD1940R	AV protocol support

Physical Layer Controllers

Manufacturer	Part Number	Description
Fujitsu Microelectronics	MB8661x	Combined link/PHY core & ICs
IBM	IBM21S85xPFD	400Mbps 1- and 3-port devices
	IBM21S760PFD	200Mbps 1- and 3-port devices
Innovative Semiconductor	SL75x	Physical layer cores
Philips Semiconductor	PDI1394P11	Physical layer IC
MacroDesigns	-	Physical layer cores
Phoenix Technologies	VirtualLink	100, 200, and 400Mbps 1394a-compatible cores
Sand	1394 CPHY	1394 cable physical layer core
Symbios	SYM13FW403	1394 cable PHY interface IC
NEC	uPD72850	3-port PHY IC
Texas Instruments	TSB11C01	Up to 400Mbps PHY ICs
	TSB11LV01	
	TSB14C01A	
	TSB21LV03A	
	TSB41LV0x	
Sony	CXD1944R	3-Port 200Mbps PHY IC

Enabling the convergence

The IEEE 1394 protocol, USB, Ethernet, and IrDA will be the data channels of the future. Any embedded system that needs to share information (and what systems won't?) will use at least one of the aforementioned communication methods. IEEE 1394 provides the highest throughput, as well as providing isochronous capability and peer-to-peer support. These features make it a prime candidate as the driver for the consumer, computer, and communications convergence. Proposed enhancements and additions to the protocol include targeting higher speeds, home networking, fiber transmission, and wireless IR transmission. As more devices support 1394, the prices for silicon will continue to drop rapidly, which will in turn cause more engineers to design in this protocol. **esp**

John Canosa is the manager of the Multimedia and Imaging Practice of Questra Corp., where he is responsible for the group that designs and develops hardware and software for devices targeting the multimedia and digital imaging markets. You can reach him electronically at jcanosa@questra.com.

References & Useful Sites

1. Anderson, Don, Mindshare, Inc. *FireWire System Architecture: IEEE 1394*. Reading, MA: Addison-Wesley, 1998.
2. *IEEE 1394-1995 Serial Bus Specification*
3. *ISO/IEC 13213 (ANSI / IEEE 1212) CSR Architecture Specification*
4. *www.1394ta.org*, The 1394 Trade Association home page.
5. *www.microsoft.com/hwdev/busbios/1394support.htm*, Microsoft's 1394 support.
6. *ftp://ftp.austin.ibm.com/pub/chrptech/1394ohci*, The 1394 OHCI Specification.
7. *www.ti.com/sc/docs/msp/1394/1394.htm*